Implementation: Target Architectures

- different target architectures for numerical simulations:
  - **monoprocessors**: many everyday simulation applications (like control, e.g.) are designed to run on PCs or ordinary workstations; obtaining optimum efficiency requires knowledge of how modern microprocessors work
  - **supercomputers**: numerical simulations have always been the most important application of high-performance computers, as well as the driving force of supercomputer development; obtaining optimum efficiency requires architecture-based tuning
- computer development follows Moore's law: every 5 years a performance increase by 10 (for both classes)
- performance distance between mass market computers and supercomputers nearly constant (factor >100)

Modern Microprocessors

- obvious trends:
  - increasing clock rates (> 1GHz almost standard)
  - more MIPS, more FLOPS
  - very-, ultra-, and ???-large scale integration; hence, more transistors and more functionality on the chip
  - longer words: 64 Bit architectures are standard (workstations) or coming (PCs)
- important features:
  - **RISC** (Reduced Instruction Set Computer) technology
  - well-developed **pipelining**
  - **superscalar** processor organization
  - **caching** and multi-level memory hierarchy
  - **VLIW**, **Multi Thread** Architecture, On-chip multiprocessors, ...

RISC Technology, Pipelining

- RISC-technology:
  - counter-trend to CISC: more and more complex instructions entailing *microprogramming*
  - now instead:
    - relatively small number of instructions (tens)
    - simple machine instructions, fixed format, few address modes
    - one cycle per instruction
    - *load-and-store* principle: only explicit LOAD/WRITE instructions have memory access
    - no more need for microprogramming
- pipelining:
  - decompose instructions into simple steps involving different parts of the CPU: load, decode, reserve registers, execute, write results (Alpha 21164, FP-DIV double prec.: 61 clocks)

Pipelining, Superscalar Processors

- pipelining (continued):
  - further improvement: reorder steps of an instruction (LOAD as early as possible, WRITE as late as possible: avoids risk of idle waiting time)
  - best case: identical instructions to be pipelined/overlapped, as in *vector processors*
  - pipelining needs different functional units in the CPU that can deal with the different steps in parallel; therefore:
- superscalar processor organization:
  - several parts of the CPU are available in more than 1 copy
  - example: MIPS R10000 has 5 execution pipelines
    - one for FP-multiplication, one for FP-addition
    - two integer ALU (arithmetic-logical units)
    - one address pipeline

Cache Memory

- cache memory:
  - aim: reduce memory access time / latency (CPU performance increased faster than memory access speed)
  - breakthrough: IBM 360/370
  - cache memory: small and fast on-chip memory, keeps part of the main memory
  - optimum: needed data is always available in cache memory
  - access time of main memory / cache / effective access time, hit probability p: $t_e = p \cdot t_c + (1-p) \cdot t_m$
  - look for strategies to ensure p close to 1:
    - choice of section: what to be kept in cache?
    - ensure *locality* of data (instructions in cache need data in cache)
    - strategies for fetching, replacement, and updating
    - association: how to check whether data are available in cache?
    - consistency: no different versions in cache and main memory

Memory Hierarchy

- today: several cache levels
  - SGI Power Challenge: 32 kB on-chip *primary cache*, up to 16 MB off-chip *secondary cache*; sometimes also level-3 cache
  - together: **memory hierarchy**: register, (level-1/2/3) cache, main memory, hard disk, remote memory: the faster, the smaller
  - notion of the target computer's memory hierarchy is important for numerical algorithms' efficiency:
    - example: matrix-vector product Ax with A too large for cache
    - standard algorithm: outer loop over rows of A, inner loop for scalar product of one row of A with x
    - if current contents of cache are some rows of A, it's OK
    - if current contents of cache are some columns of A: slow!
    - tuning crucial: peak performance up to 4 orders of magnitude higher than performance observed in practice (without tuning)

Parallelization, Parallelism, Parallel Computers

- parallel computers – distributed systems: frontier?
- what has to be distinguished:
  - *What* is to be parallelized (code or data; competition)?
  - *How* is parallelization done (concurrent / overlapping; control-driven / data-driven)?
  - *Where* is parallelization done (programs / processes / machine instructions / microinstructions)?
  - *Who* parallelizes (manual or explicit / interactive / automatic or implicit)?
  - *topology* of the system: arrangement of processors, structure of network, static or dynamic topology)
  - *synchronization:* loose or tight coupling
  - *communication:* implicitly via shared memory or explicitly via messages

Topologies

- different possibilities of arrangement:
  - **static network topologies**:
    - bus, ring, grid, or torus
    - binary tree or fat tree
    - hypercube
  - **dynamic network topologies**:
    - crossbar switch
    - shuffle exchange network
- crucial quantities:
  - diameter (longest path between two processors)
  - number of network connections (ports) per processor
  - parallel communications possible?
  - existence of bottlenecks?

Flynn's Classification (1972)

- SISD: *Single Instruction Single Data*
  - classical von-Neumann monoprocessor
- SIMD: *Single Instruction Multiple Data*
  - **vector computers**: extreme pipeling, one instruction applied to a sequence (vector) of data (CRAY 1,2,X,Y,J/C/T90,...)
  - **array computers**: array of processors, concurrency (Thinking Machines CM-2, MasPar MP-1, MP-2)
- MIMD: *Multiple Instruction Multiple Data*
  - **multiprocessors**: *distributed memory* (loose coupling, explicit communication; Intel Paragon, IBM SP-2) or *shared memory* (tight coupling, global address space, implicit communication; most workstation servers) or *nets/clusters*
- MISD: *Multiple Instruction Single Data* : rare

Memory Access Classification

- other criteria: *scalability (S), programming model (PM), portability (P),* and *load distribution (L)*
- UMA: *Uniform Memory Access*
  - shared memory systems: SMP (symmetric multiprocessors, parallel vector processors); PC- and WS-servers, CRAY YMP
  - advantage: P, PM, L; drawback: S
- NORMA: *No Remote Memory Access*
  - distributed memory systems; clusters, IBM SP-2, iPSC/860
  - advantage: S; drawback: P, PM, L
- NUMA: *Non-Uniform Memory Access*
  - systems with virtually shared memory; KSR-1, CRAY T3D/T3E, CONVEX SPP
  - Advantage: PM, S, P; drawback: cache-coherence, commun.